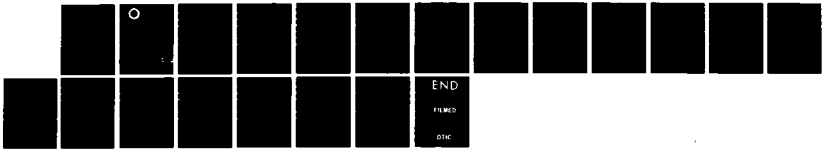
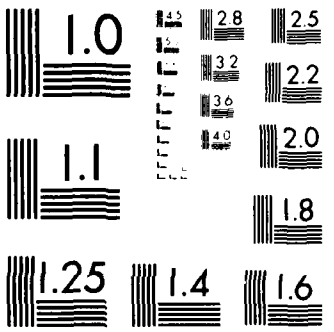
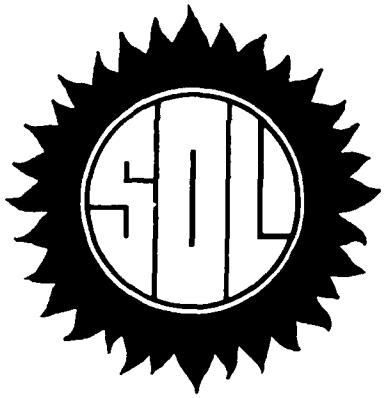
AD-A151 480 SOFTWARE AND ITS RELATIONSHIP TO METHODS(U) STANFORD UNIV CA DEPT OF OPERATIONS RESEARCH P E GILL ET AL. NOV 84 SOL-84-10 ARO-21592.4-MA N00014-75-C-0267 1/1

UNCLASSIFIED F/G 9/2 NL







Systems
Optimization
Laboratory

SOFTWARE AND ITS RELATIONSHIP TO METHODS†

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 84-10

November 1984

Department of Operations Research
Stanford University
Stanford, CA 94305

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

SOFTWARE AND ITS RELATIONSHIP TO METHODS[†]

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 84-10

November 1984

Research and reproduction of this report were partially supported by National Science Foundation Grants MCS-7926009 and ECS-8312142; Department of Energy Contract DE-AM03-76SF00326, PA# DE-AT03-76ER72018; Office of Naval Research Contract N00014-75-C-0267; and Army Research Office Contract DAAG29-84-K-0156.

[†]Presented as an invited paper at the SIAM Conference on Numerical Optimization, June 12-14, 1984, Boulder, Colorado.

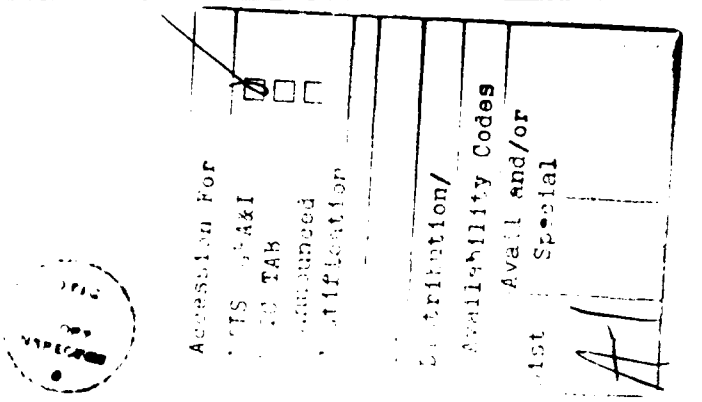
SOFTWARE AND ITS RELATIONSHIP TO METHODS †

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

ABSTRACT

One view of numerical software is that it is simply a computer implementation of a known method. Implicit in this view is the assumption that the flow of information is in one direction only. However, developments in methods and software are intimately related, and neither is complete if considered in isolation. In this paper, we illustrate how the development of numerical software has influenced our research in optimization methods.



This research was supported by the U.S. Department of Energy Contract DE-AM03-76SF00326, PA No. DE-AT03-76ER72018; National Science Foundation Grants MCS-7926009 and ECS-8312142; the Office of Naval Research Contract N00014-75-C-0267; and the U.S. Army Research Office Contract DAAG29-84-K-0156.

† Presented as an invited paper at the SIAM Conference on Numerical Optimization, June 12-14, 1984, Boulder, Colorado.

1. Introduction

It is almost a truism that an optimization method cannot be treated as practical unless an implementation has been produced and a significant amount of computation performed. Thus, research on optimization methods necessarily overlaps heavily with the development of software. Since the advent of serious work on numerical software, much has been written about the complexities that arise when transforming any mathematical algorithm into an implementation (see, e.g., Cody, 1974; Gill *et al.*, 1979; Cowell, 1983). Although most workers in optimization are aware of such issues, the effect of implementation on methods is much less widely understood. In fact, the relationship between methods and software is sometimes described simply by defining an implementation as a concrete realization of a theoretical algorithm.

In our view, this statement does not include the crucial influence that implementation may have on theoretical algorithms. Implementation must always be considered by the algorithm designer in the sense that the steps of a theoretical method should be implementable. However, our experience suggests that implementation has a much more substantive effect on methods. This paper accordingly develops the theme that *software creates new methods*, and that a method can be produced in its most effective form only in conjunction with a careful implementation. In order to avoid vague generalities, our own experiences with specific issues will be cited to illustrate the sometimes subtle interconnections that can occur. Thus, we shall describe the *evolution* of certain methods as a result of implementation.

Initialization and *communication* are two critical areas in which the process of implementation influences an algorithm. In describing the computation associated with an algorithm, the basic iteration is usually the main concern. In implementing a method, however, initialization is crucial — not only the computation that must be performed to initiate the method, but also the information that is communicated to and from the algorithm. In our experience, the nature, cost and difficulty of initialization procedures may be obscured by the purely mathematical description of an algorithm. Furthermore, the representation of the information needed by an algorithm — e.g., a particular matrix factorization — is often left undefined until required by an implementation. This latter tendency sometimes leads to a surprisingly optimistic belief in "black box" software; we shall describe several examples in which the use of "off-the-shelf" codes in a complex algorithm introduces substantial inefficiencies because of poor communication between software modules.

A careful implementation also illuminates questions of detail that are typically ignored in a purely theoretical setting. In the analysis of algorithms, a "good" proof should have the widest possible application and generality. In order to facilitate the construction of such proofs, methods are often described with a comparable level of abstraction. For example, theoretical descriptions of active-set nonlinear programming methods typically treat linear and nonlinear constraints in a uniform way. Consequently, it is not always appreciated that substantial improvements in performance can result when algorithms exploit the different properties of linear and nonlinear constraints. The scope for increased efficiency is not restricted to savings such as avoiding the recomputation of gradients for linear constraints. Linear constraints have the important property that, from any non-optimal feasible point, directions can be generated along which there must exist *intervals* of feasible points. This property allows the development of methods that always retain feasibility with respect to the linear constraints. For such methods, the path to the solution and the number of iterations tend to differ completely from those of methods that ignore linearity. The many non-obvious effects of separate treatment of linear and nonlinear constraints will be a recurrent topic in this paper.

2. Background on quadratic and nonlinear programming

2.1. Quadratic programming. Many of our examples will be drawn from a relatively simple optimization problem – quadratic programming (QP):

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x + \tfrac{1}{2} x^T H x \\ \text{subject to} \quad & \ell \leq \begin{Bmatrix} x \\ Ax \end{Bmatrix} \leq u, \end{aligned}$$

where H is symmetric. The constraints involving the matrix A will be called the *general* constraints; the remaining constraints will be called *bounds*. An *equality* constraint corresponds to setting $\ell_i = u_i$. Similarly, a special "infinite" value for ℓ_i or u_i is used to indicate the absence of one of the bounds.

In general, an iterative process is required to solve a quadratic program. (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Each new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{1}$$

where the *step length* α is a non-negative scalar, and p is called the *search direction*.

Throughout this paper, we shall consider only a particular *active-set feasible-point method* for quadratic programming (see Gill *et al.*, 1984*a*). An important feature of the method is that, once any iterate is feasible, all subsequent iterates remain feasible.

The essence of the method is the definition of a *working set* of constraints (general and bound) that are satisfied exactly at x. The search direction is constructed so that the constraints in the working set remain *unaltered* for any value of the step length. For a bound constraint in the working set, this property is achieved by setting the corresponding component of the search direction to zero. Thus, the associated variable is *fixed*, and specification of the working set induces a partition of x into *fixed* and *free* variables. During a given iteration, the fixed variables are effectively removed from the problem; since the relevant components of the search direction are zero, the columns of A corresponding to fixed variables may be ignored.

Let m denote the number of general constraints in the working set and let n_{FR} denote the number of free variables. Let C denote the $m \times n_{FR}$ submatrix of general constraints in the working set corresponding to the free variables, and let p denote the search direction *with respect to the free variables only*. The general constraints in the working set will be unaltered by any move along p if

$$Cp = 0. \tag{2}$$

In order to compute p, the *TQ factorization* of C is used:

$$CQ = (0 \quad T), \tag{3}$$

where T is an $m \times m$ reverse-triangular matrix (i.e., $t_{ij} = 0$ if $i + j < m$), and the non-singular matrix Q is the product of either orthogonal or stabilized elementary transformations (see Gill *et al.*, 1984*c*). In this paper, we consider only the case where Q is orthogonal. If T is non-singular and the columns of Q are partitioned so that

$$Q = (Z \quad Y), \tag{4}$$

where Y is $n_{FR} \times m$, then the $(n_{FR} - m)$ columns of Z form a basis for the null space of C. Thus, p will satisfy (2) only if

$$p = Zp_z \tag{5}$$

for some vector p_z.

The definition of p_z in (5) depends on whether the current point is feasible. If not, p_z is taken as $-Z^Tq$, where q is the gradient of the current sum of infeasibilities with respect to the free variables. Otherwise, p_z is the solution of

$$R_z^T R_z p_z = -Z^T q, \tag{6}$$

where R_z is upper triangular and q is the gradient of the quadratic objective function with respect to the free variables. The Cholesky factor R_z is closely related to the projection of the Hessian matrix (with respect to the free variables) into the subspace defined by Z.

At each iteration, the working set is changed by adding or deleting constraints. Each change in the working set leads to a simple change to C: if the status of a general constraint changes, a *row* of C is altered; if a bound constraint enters or leaves the working set, a *column* of C changes. The method recurs explicit representations of T, Q, R_z and q_Q (the vector Q^Tq, where q ($\equiv c + Hx$) is the gradient of the quadratic objective function).

2.2. Nonlinear programming. The second problem to be discussed is the *nonlinear programming problem*:

$$\text{NP} \qquad \underset{x \in \Re^n}{\text{minimize}} \quad F(x)$$

$$\text{subject to} \quad \ell \leq \begin{Bmatrix} x \\ A_L x \\ c(x) \end{Bmatrix} \leq u,$$

where $F(x)$ (the *objective function*) is a smooth nonlinear function, A_L is a constant matrix of linear constraints, and $c(x)$ is a vector of smooth nonlinear constraint functions. Note that simple bounds and linear constraints are represented *separately* from nonlinear constraints.

It is widely considered today that sequential quadratic programming (SQP) methods are the most effective general techniques for treating constraint nonlinearities. The idea of solving nonlinear programs by a sequence of quadratic programming subproblems was first suggested by Wilson (1963). SQP methods were popularized mainly by Biggs (1972), Han (1976) and Powell (1977). (A brief history of SQP methods and an extensive bibliography are given in Gill, Murray and Wright, 1981. For a survey of recent results and references, see Powell, 1983.)

The basic structure of an SQP method involves *major* and *minor* iterations. Associated with each major iteration are a search direction p, a set of multipliers μ for the nonlinear constraints, and a working set of nonlinear constraints (a prediction of the constraints that are satisfied exactly at the solution). In the methods considered in this paper, all of these quantities are computed from a quadratic programming subproblem. The vector p of (1) is the solution itself, μ is the Lagrange multiplier vector from the QP subproblem, and the "nonlinear" working set is the final active set of the subproblem. Since solving such a subproblem is itself an iterative procedure, the *minor* iterations of an SQP method are those of the QP method.

The QP subproblem is defined by a set of linear constraints and a quadratic objective function. In most SQP methods, the linear constraints of the subproblem are linearizations of the original constraints about the current point. In order to attain rapid convergence, the objective function of the subproblem must approximate the Lagrangian function. Each major iteration thus includes a quadratic programming subproblem of the form

$$\underset{p}{\text{minimize}} \quad g^Tp + \tfrac{1}{2}p^THp$$

$$\text{subject to} \quad \bar{\ell} \leq \begin{Bmatrix} p \\ A_L p \\ A_N p \end{Bmatrix} \leq \bar{u},$$

where g is the gradient of F at x, the matrix H is an approximation to the Hessian of the Lagrangian function, and A_N is the Jacobian matrix of $c(x)$ evaluated at x. Let ℓ in NP be partitioned into three sections -- ℓ_B, ℓ_L and ℓ_N -- corresponding to the bound, linear and nonlinear constraints. The vector $\bar{\ell}$ is similarly partitioned, and is defined as

$$\bar{\ell}_B = \ell_B - x, \quad \bar{\ell}_L = \ell_L - A_L x, \quad \text{and} \quad \bar{\ell}_N = \ell_N - c,$$

where c is the vector of nonlinear constraints evaluated at x. The vector $\bar{u}$ is defined in an analogous fashion.

Having obtained p, the major iteration proceeds by determining a steplength α in (1) that produces a "sufficient decrease" in some *merit function* (a combination of the objective and constraint functions that measures progress toward the solution).

Because exact second derivatives may be unavailable in practice, most work on SQP methods has concentrated on the case in which the matrix H in the subproblem is a *positive-definite quasi-Newton* approximation to the Hessian of the Lagrangian function. (For a review of quasi-Newton methods, see Dennis and Schnabel, 1983.) After $\bar{x}$ has been obtained, the new Hessian approximation is typically defined as a low-rank modification of the old, so that *the Hessian matrices of successive quadratic programming subproblems are related in a special way* (see Section 6). With the BFGS update, for example, the new matrix $\bar{H}$ is given by

$$\bar{H} = H - \frac{1}{s^T H s} H s s^T H + \frac{1}{y_\ell^T s} y_\ell y_\ell^T, \tag{7}$$

where $s = \bar{x} - x$ (the change in x), and y_ℓ is the difference in gradients of the Lagrangian function, i.e.,

$$y_\ell = \bar{g}_\ell - g_\ell = (\bar{g} - \bar{A}_N^T \bar{\lambda}) - (g - A_N^T \lambda),$$

where $\bar{\lambda}$ and λ are Lagrange multiplier estimates at the new and old points. In most SQP methods, both $\bar{\lambda}$ and λ are set to μ, the multipliers of the latest subproblem.

It should be clear from the above description that an SQP method is inherently complex. Any implementation of an SQP method must include: (1) the solution of a quadratic programming subproblem (with some provision for the treatment of inconsistent constraints); (2) definition of a merit function; and (3) specification of the Hessian.

3. Finding an initial feasible point

As a brief example of how an algorithm description can be misleading with respect to the efficiency of the implementation, consider an SQP method in which a so-called two-phase (primal) quadratic programming method is used to solve the subproblem. The two phases are: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the quadratic objective function in the feasible region (the QP phase). When describing such a method, it is convenient to state simply that an initial feasible point must be found by a phase-1 procedure (the present authors have done so many times!). Unfortunately, this form of description may be (wrongly) interpreted as implying that a "black box" phase-1 simplex linear programming procedure should be invoked before the QP phase. If so, serious inefficiency could result if several simplex iterations were required in order to find a feasible point (since a vertex solution would need to be created); standard linear programming software is also highly unlikely to produce the factorizations needed to initiate the QP phase. Consequently, a two-phase

quadratic programming method might be considered as ineffective simply because of a lack of detail in specifying the initialization procedure.

On the contrary, the proper implementation of a two-phase quadratic programming method should reflect the essential *sameness* of the linear algebraic computations associated with iterations in both the feasibility and QP phases. In particular, each iteration involves an update of the TQ factorization of the working set. It is typical for the subproblems in later major iterations of an SQP method to reach optimality in a *single iteration* because the optimal active set is available before solving the subproblem. In this case, the phase-1 procedure merely performs a feasibility check that would be required in any case.

In our implementation, the computations in both phases are performed by exactly the same modules. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function. The feasibility phase does *not* perform the standard simplex method (i.e., it does not find a vertex), and requires *no additional work* in the later iterations of an SQP method.

4. Factorization of the working set

The quadratic programming method of Section 2.1 requires an explicit matrix Q in order to perform updates to the working set. In this section, we consider the evolution of a quadratic programming method with respect to its computation of the initial Q.

As mentioned previously, the active set of each subproblem in an SQP method eventually becomes the active set of the *nonlinear* problem. Since the iterations in a QP method are directed entirely toward finding the active set, it would appear that a good initial estimate of the active set could permit later subproblems to reach optimality in only one iteration.

Our first idea along these lines was to include a "warm start" option in the QP method, i.e., to specify the desired initial working set as input to the feasibility phase (for details, see Gill *et al.*, 1985*a*). Beyond adding this option, however, the original implementation retained in large part the philosophy that the eventual SQP method should utilize as much as possible an "off-the-shelf" version of the quadratic programming method discussed in Section 2.1.

Computation of the TQ factorization associated with a specified working set may be viewed as *updating* an existing factorization as new rows are added in the last position. Assume that the TQ factorization (3) of C is available, and consider the matrix $\bar{C}$, which is C augmented by the row c^T. Then

$$\bar{C}Q = \begin{pmatrix} C \\ c^T \end{pmatrix} Q \equiv \bar{T} = \begin{pmatrix} 0 & T \\ s^T & t^T \end{pmatrix}, \tag{8}$$

where s and t are the relevant partitions of $Q^T c$. Let $\hat{Q}$ denote a Householder matrix of the form

$$\hat{Q} = I - \frac{1}{\beta} u u^T,$$

where the vector u and scalar β are chosen to annihilate all but the last element of s, and to leave t unchanged. (For details of how these quantities are defined, see Stewart, 1973.) Then

$$\bar{T}\hat{Q} = \begin{pmatrix} 0 & 0 & T \\ 0 & \tau & t^T \end{pmatrix} \tag{9}$$

or

$$\bar{C}\hat{Q} = \begin{pmatrix} 0 & \bar{T} \end{pmatrix},$$

where $\bar{Q} = Q\tilde{Q}$.

In a general-purpose QP algorithm, the initial TQ factorization of C would probably be computed by the so-called "standard" procedure, which can be interpreted as a version of (8) and (9) in which the rows of C are added to the null matrix one by one. The initial Q matrix is taken as the identity, and the initial T is the null matrix. While computing the factorization, the sequence of Householder transformations is stored *in compact form* (i.e., Q is not stored explicitly); the vector c^TQ needed in (8) is obtained by applying the sequence of stored transformations. Once the initial factorization has been completed, the necessary explicit matrix Q is obtained by multiplying the compact Householder transformations together in reverse order.

A shift in perspective occurs when this computation is performed within an SQP method applied to a problem that contains *both linear and nonlinear* constraints. (We shall use the term *linear row* to denote a row of the working set associated with a linear constraint in the original problem, and similarly for *nonlinear row*.) Since the rows of A_L are the same in each subproblem, it seems highly desirable to avoid refactorization of linear rows. We emphasize that by considering this possibility, we have already deviated from seeking a general-purpose quadratic programming code, which is unlikely to make distinctions between row types. Hence, to take advantage of the presence of linear constraints, a more specialized QP method must be devised.

Even if the QP method can distinguish among rows, it is still not straightforward to exploit constraint linearities in computing the initial TQ factorization. In order for the same Householder transformations to be used again, the *order* of the rows must remain unchanged. However, the order of the rows in the final C is determined by the order in which constraints enter or leave the working set during the iterations of the quadratic programming method. If the constraints of the nonlinear problem are ordered so that the linear constraints precede the nonlinear constraints, and if constraints are always added to the initial working set in order, we can determine in the major iteration how many rows of C have not changed since the previous subproblem. Ideally, the part of the TQ factorization corresponding to the unchanged linear rows need not be recomputed. In particular, if the first m_L rows of C are linear, the implementation should be able to utilize the first m_L Householder transformations from the initial TQ factorization of the previous subproblem. Eventually, as the SQP iterates converge, the active set will not change between subproblems, and all the Householder transformations corresponding to linear rows can be saved.

Unfortunately, this aim is extremely difficult to achieve within an implementation that uses the standard procedure to obtain the initial TQ factorization. The explicit Q from the previous subproblem does *not* allow reconstruction of the Householder transformations from which it was computed. Therefore, in order to avoid refactorizing the linear rows, the QP method would have to *save* the compact transformations corresponding to the linear constraints at the beginning of the initial working set. Even then, the saved compact transformations would be of no use if just *one* bound had changed status during the QP iterations. (The column dimension of C changes when the set of free variables is altered.)

Although this analysis is discouraging, we persisted in trying to exploit constraint linearities in the initial TQ factorization because of the large number of practical problems with a significant number of linear constraints and bounds. The result was to change the definition of both the QP and the SQP methods. Rather than attempt to define some way for the quadratic programming method to distinguish among row types, computation of the initial TQ factorization was removed from the QP method proper to the major iteration, which "naturally" has access to information about which constraints are linear and nonlinear. A necessary consequence was that the quadratic programming method must include a *hot start* option, in which the input parameters of the quadratic program include not only the specification of a working set, but also its associated TQ

factorization.

In addition to this change in structure of the quadratic programming method, the procedure for computing the initial factorization was changed. In the *updating method*, the explicit matrix Q *from the previous subproblem* is taken as the initial matrix Q in (8). In contrast to the standard procedure, where the initial Q is taken as the identity matrix, this means that each new row must be transformed by a full orthogonal matrix rather than a sequence of Householder transformations, and that each Householder transformation must be multiplied into Q after the corresponding row has been transformed. The benefit from this procedure with respect to linear rows of C is that the final matrix Q from the previous subproblem automatically transforms the initial constant rows of C into reverse-triangular form (regardless of any changes that occurred in the status of bound constraints). Thus, no work is required to obtain the first m_L rows of T.

This example of the shift in a method because of implementation is highly relevant because of the complex issues that led to our choice of the updating method. It should be emphasized that the updating method can be *less expensive* than the combination of the standard method with the special formation of Q, *depending on the number of linear rows*. (For detailed operation counts, see Gill *et al.*, 1985*b*.)

5. The initial projected Hessian for indefinite quadratic programs

Our quadratic programming algorithm does not require H to be positive definite, and hence must allow for possible indefiniteness and singularity of the projected Hessian in the QP phase. (The quadratic programming subproblems in an SQP method may be indefinite when exact second derivatives are used to define the Hessian.) The treatment of indefiniteness depends critically on the result that, if the *initial* projected Hessian is positive definite, the projected Hessian can thereafter become *indefinite only when a constraint is deleted from* the working set (see Gill and Murray, 1978). This property implies that indefiniteness can be controlled explicitly, and permits a numerically stable Cholesky factorization to be used in solving (6) for p_Z, even when the projected Hessian is indefinite. (For details, see Gill and Murray, 1978, and Gill *et al.*, 1985*a*.)

In an implementation, the initialization procedure must therefore determine a positive-definite projected Hessian. How can this be achieved most effectively? We consider several strategies for initializing the projected Hessian, all of which are based on the observation that the projected Hessian matrix will be positive definite if enough constraints are included in the initial working set. (The null matrix is positive definite by definition, corresponding to the case when C contains n_{FR} constraints.) This suggests somehow *adding constraints to the working set* to make the projected Hessian positive definite.

The first strategy deals with indefiniteness through the initialization of the feasibility phase. The initial working set in the phase-1 procedure is constructed to contain n_{FR} constraints (i.e., to define a vertex) by augmenting the "natural" working set (the constraints exactly or nearly satisfied at the starting point) with a suitable number of "temporary" bounds, each of which has the effect of temporarily fixing a variable at its current value. Since the phase-1 procedure is equivalent to the simplex method if started at a vertex, the final working set of the feasibility phase will also be a vertex. Hence, the initial Z of the QP phase will be null, and the projected Hessian will be (trivially) positive definite.

A second option is to carry out a phase-1 procedure that does not require creation of a vertex, and to form the projected Hessian as soon as a feasible point is found, using the final Z of the feasibility phase. If indefiniteness is detected while performing the Cholesky factorization of the projected Hessian, the current point can be made into a "vertex" by adding temporary bounds as described above, along with updating the TQ factorization.

With either of these techniques, the working set for the initial quadratic programming iteration is given by the square matrix

$$\bar{C} = \begin{pmatrix} C \\ E \end{pmatrix}, \tag{10}$$

where E includes rows of the identity matrix corresponding to the temporary bounds. In subsequent iterations, a temporary bound is treated as a standard constraint until it is deleted from the working set, in which case it will never be added again.

Because there seemed to be no absolute theoretical grounds for choosing between the above approaches, our initial implementation of the quadratic programming algorithm used the first strategy, for two reasons. When computing the TQ factorization from scratch, adding bounds is "free", and the size of the first TQ factorization that must be computed decreases as more bounds are included in the initial working set (for detailed analysis of the operations required, see Gill *et al.*, 1984*c*): computing the initial TQ factorization of phase 1 is cheaper with the first strategy. Furthermore, the second approach has the disadvantage that all the work of the initial Cholesky factorization in the QP phase might be wasted, depending on the (unknown) probability that the projected Hessian is positive definite at the first feasible point.

The resulting implementation of the initialization performed exactly as intended. However, a close analysis of the computation associated with each quadratic programming subproblem *within an SQP method* revealed that the creation of a temporary vertex at the beginning of phase 1 is less than satisfactory. To see why, assume that the initial projected Hessian is positive definite. In this case, each of the temporary bounds must be deleted in turn, followed by minimization of the quadratic function within a subspace of increased dimension; the effect of these moves is implicitly to compute the Cholesky factor of the projected Hessian by rows. In the extreme case of a quadratic program with an *unconstrained* solution, the work required to optimize the quadratic function on n temporary manifolds is more than twice the work required to compute the $n \times n$ Cholesky factor directly.

Although one might argue that a successful algorithm should not be distorted to cater for this special case, further examination revealed that inefficiency would be the rule rather than the exception. As an SQP method proceeds, the active sets of QP subproblems in later major iterations become the same — the set of constraints active at the solution of the original nonlinear problem (see Robinson, 1971, for a proof). Therefore, when solving the sequence of quadratic programs that arise within an SQP method, there is a high probability that the active set from one subproblem is also the correct active set for the next. In this context, the initial projected Hessian is likely to be positive definite, and creating the temporary vertex is unnecessary. When the initial working set happens to be optimal, the process of creating and then eliminating the temporary bounds requires approximately twice the work that would be required to compute the Cholesky factor directly. Furthermore, computing a smaller initial TQ factorization with temporary bounds does not lead to an *overall* saving of work, since the factorization must eventually be updated to reflect the deletion of the unnecessary bounds.

Given this observation, it might appear that the second strategy mentioned above would be revived. However, the following, even better, strategy was then developed. Rather than adding bound constraints to the working set to create a temporary vertex, the new strategy adds as many *general* constraints as necessary to ensure a positive-definite projected Hessian, but requires *no further work to update the TQ factorization or to compute the projected Hessian*. The computation proceeds as follows. At the beginning of the QP phase, the working set C and its TQ factorization (3) are available. The matrix $Z^T H Z$ is formed, and the Cholesky procedure *with symmetric interchanges* is initiated. Recall that the Cholesky procedure without interchanges

will break down if the matrix is not positive definite. However, by performing interchanges (such that the column with largest positive diagonal element is processed next at each step), we can identify the largest possible positive-definite principal minor.

In algebraic terms, assume that a permutation matrix P has been chosen so that the upper left submatrix of $P^T Z^T H Z P$ is positive definite. In effect, the columns of ZP are partitioned as $ZP = (Z_1 \;\; Z_2)$, such that $Z_1^T H Z_1$ is positive definite, i.e.,

$$Z_1^T H Z_1 = R_z^T R_z.$$

A working set for which Z_1 defines the null space can be obtained by including *the rows of* Z_2^T as temporary *general* constraints. After P is determined (by the Cholesky procedure), the columns of Z are reordered (i.e., Z is replaced by ZP); note that the properties of Z as a basis for the null space of C are unaffected by its column ordering. The minimization of the quadratic function then proceeds within the subspace defined by Z_1.

We discuss here only the case when Q is orthogonal. (For details about the case when Q is a product of stabilized elementary transformations, see Gill *et al.*, 1985*a*.) In contrast to (10), the temporarily augmented working set is given by

$$\bar{C} = \begin{pmatrix} C \\ Z_2^T \end{pmatrix}, \tag{11}$$

so that p will satisfy $Cp = 0$ *and* $Z_2^T p = 0$. By definition of the TQ factorization, $\bar{C}$ *automatically* satisfies the following:

$$\bar{C}Q = \begin{pmatrix} C \\ Z_2^T \end{pmatrix} Q = \begin{pmatrix} C \\ Z_2^T \end{pmatrix} (Z_1 \quad Z_2 \quad Y) = (0 \quad \bar{T}),$$

where

$$\bar{T} = \begin{pmatrix} 0 & T \\ I & 0 \end{pmatrix},$$

and hence the TQ factorization of (11) is free.

The implementation of this procedure involves several subtle points. The matrix Z_2 need *not* be kept fixed at its initial value, since the role of the extra constraints is purely to define an appropriate null space, the TQ factorization can therefore be updated in the normal fashion as the iterations of the quadratic programming method proceed. No work is required to "delete" the temporary constraints associated with Z_2 when $Z_1^T g = 0$, since this simply involves repartitioning Q. When deciding which constraint to delete, the multiplier vector associated with the rows of Z_2^T is given by $Z_2^T g$, and the multipliers corresponding to the rows of the "true" working set C are the least-squares multipliers that would be obtained if the temporary constraints were not present (see Gill *et al.*, 1985*a*).

Although one might claim that the final method could have been developed independently of any software, we believe that it is unlikely that all the ramifications of each choice could have been fully understood without careful implementation and detailed *a posteriori* analysis of the computation.

6. Representation of the Hessian

Section 4 has described the changes in representation of a factorization of the working set in our quadratic programming method because of its intended use within an SQP method. We now

turn to an even more specialized question: the representation of the Hessian matrix in a *quasi-Newton* SQP method, i.e., one in which H is a *positive-definite* approximation to the Hessian of the Lagrangian function that is modified by a low-rank change between subproblems.

In all the quadratic programming methods discussed thus far, the Cholesky factorization of the initial projected Hessian is computed from scratch at the beginning of the QP phase. Although this procedure is perfectly satisfactory for a single quadratic program, it has certain disadvantages in a quasi-Newton SQP method. In particular, when applied to a problem with no constraints or only linear constraints, such a method is much less efficient than standard quasi-Newton methods for these problems, in which the Hessian approximation is *updated* between iterations. The question therefore arises: can an SQP method maintain an efficient treatment of nonlinear constraints, yet remain competitive with unconstrained or linearly constrained quasi-Newton methods if constraint nonlinearities are not present?

Such efficiency can be guaranteed if an algorithm satisfies a specific criterion: when the working set includes m_N nonlinear rows, the initialization of T, Q and R_Z should require $O\left(m_N^2 + m_N^2 n_{FR}\right)$ operations. This criterion is *not* satisfied by the hot-start option described in Section 4, since changes in Z mean that both $Z^T H Z$ and R_Z must be computed from scratch. Our approach involves recurring R_Q, the Cholesky factor of the *transformed* Hessian approximation $Q^T H Q$ ($\equiv H_Q$) in *both the major and minor iterations*. (The form (4) of Q implies that the matrix R_Z needed to compute the search direction is simply the upper left corner of R_Q.)

To illustrate how the method works, consider the case when the working set at a given iteration contains m_L linear rows and a *single* nonlinear row. Assume that, on completion of the QP subproblem at the point x, R_Q is available. As indicated by (8) (9), the effect of replacing the last row of the working set is to post-multiply Q (and therefore R_Q) by a matrix of rank one. Since $\bar{Q} = Q\tilde{Q}$, where $\tilde{Q} = I - (1/\beta)uu^T$, we have

$$R_Q \tilde{Q} \equiv \tilde{R}_Q = R_Q - \frac{1}{\beta} R_Q u u^T. \tag{12}$$

The new Cholesky factor $\bar{R}_Q$ is then found by constructing an orthogonal matrix P that restores upper-triangular form to $\tilde{R}_Q$, i.e.,

$$\bar{R}_Q = P\tilde{R}_Q.$$

A suitable matrix P can be constructed from two sweeps of plane rotations; for more details, see Gill *et al.* (1974). In general, if the working set contains m_N nonlinear rows, m_N rank-one updates must be applied to obtain $\bar{R}_Q$.

Given this procedure for updating R_Q, we now show how to recur the transformed gradient q_Q, which changes in two different situations. First, as constraints enter or leave the working set, the plane rotations used to update Q can simply be applied to q_Q. The second change in q_Q occurs when p is replaced by $\bar{p} = p + \alpha\delta p$, where the search direction δp is defined as

$$\delta p = Z\delta p_Z = Q\begin{pmatrix} \delta p_Z \\ 0 \end{pmatrix}. \tag{13}$$

Let $\bar{q}_Q$ denote the transformed gradient at $\bar{p}$. It follows from (13) and the definitions of q and R_Q that

$$\begin{aligned} \bar{q}_Q &= Q^T c + Q^T H(p + \alpha\delta p) \\ &= Q^T(c + Hp) + \alpha Q^T H \delta p \end{aligned}$$

$$= q_Q + \alpha Q^T H Q \begin{pmatrix} \delta p_z \\ 0 \end{pmatrix}$$

$$= q_Q + \alpha R_Q^T R_Q \begin{pmatrix} \delta p_z \\ 0 \end{pmatrix},$$

which shows that H is not required to update q_Q.

In order to avoid access to H in the quasi-Newton update between subproblems, R_Q can be updated directly. In this situation, Q remains unchanged, and H undergoes a rank-two modification. The BFGS update (7) of H leads to the following change in H_Q:

$$\bar{H}_Q = H_Q - \frac{1}{s_Q^T H_Q s_Q} H_Q s_Q s_Q^T H_Q + \frac{1}{y_Q^T s_Q} y_Q y_Q^T, \tag{14}$$

where $y_Q = Q^T(\bar{g}_\ell - g_\ell)$, $s_Q = Q^T(\bar{x} - x)$ and g_ℓ denotes the gradient of the Lagrangian function. This update may be expressed as a *rank-one* update to R_Q (see Dennis and Schnabel, 1981). Let σ and γ denote the scalars $(s_Q^T H_Q s_Q)^{\frac{1}{2}}$ and $(y_Q^T s_Q)^{\frac{1}{2}}$ respectively. The updated matrix (14) may then be written as $\bar{H}_Q = \tilde{R}_Q^T \tilde{R}_Q$, where

$$\tilde{R}_Q = R_Q + vw^T, \quad \text{with} \quad v = \frac{1}{\sigma} R_Q s_Q, \quad w = \frac{1}{\gamma} y_Q - \frac{1}{\sigma} H_Q s_Q.$$

Again, the matrix $\tilde{R}_Q$ may be restored to upper-triangular form by two sweeps of plane rotations.

It should be emphasized that these changes in the representation of the projected Hessian imply that the matrix R_Q must be updated during the feasibility phase as well as the QP phase. The SQP method is also altered in a fundamental way. Now, the factor R_Q is altered by three sources: changes in the constraint gradients (see (12)); changes in the curvature of the Lagrangian function (the quasi-Newton update); and, finally, changes in the prediction of the active set.

Several interesting research issues have resulted from these changes in the SQP method. If the correct q-superlinear convergence rate is to be achieved at the solution, it is necessary to show that small changes in the variables lead to small changes in the matrix R_Q (for details of the proof, see Gill *et al.*, 1985*b*). Similarly, certain choices of least-squares multiplier estimates lead to methods similar to projected quasi-Newton methods (see, e.g., Murray and Wright, 1978; Coleman and Conn, 1982; Gabay, 1982; and Nocedal and Overton, 1983; Byrd and Schnabel, 1984). In the next section, we shall discuss a class of new methods with a more specialized treatment of linear constraints.

7. A more specialized treatment of linear constraints

The ability to update an approximation to the Hessian of the Lagrangian function as each new nonlinear row is factorized leads to a class of methods with *separate active-set strategies for the linear and nonlinear constraints*. Instead of using the quadratic programming subproblem to define the complete working set, the "active" linear constraints and bounds are determined in the *major* iteration by an active-set strategy typical in methods for linearly constrained optimization (see Gill and Murray, 1974; Gill, Murray and Wright, 1981). With this approach, the linear rows never need to be refactorized even during early major iterations, before the active linear constraints have been determined.

In such a method, an initial point is found that is feasible with respect to the linear constraints and bounds (see Section 3). Within each major iteration, a working set of bounds and linear constraints is defined in the usual way. Let C_L denote the submatrix of general linear constraints in the working set corresponding to the free variables, with Z_L the corresponding basis for the null space. The search direction is then determined from the modified quadratic programming subproblem

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & g^T p + \tfrac{1}{2} p^T H p \\ \text{subject to} \quad & \bar{\ell} \leq A_N p \leq \bar{u}, \\ & C_L p = 0, \end{aligned}$$

where A_N denotes the columns of the Jacobian of nonlinear constraints corresponding to the free variables.

During the major iterations, the TQ factorization of C_L is recurred as linear constraints enter and leave the working set. These changes are reflected in the Cholesky factor R_L of $Z_L^T H Z_L$, and are identical to the updates that occur in an active-set method for linearly constrained minimization — i.e., R_L grows by a row and column when a constraint leaves the working set, R_L shrinks by a row and column when a constraint enters the working set. Before solving the QP subproblem, each nonlinear row is added to the overall working set, leading to updates to the TQ factorization and to R_L, as in (12). (The matrix R_Z required to compute the quadratic programming search direction is in the upper left-hand corner of R_L.) On completion of the subproblem, R_L is updated directly to incorporate new curvature of the Lagrangian function.

If the working set contains no nonlinear constraints, the method becomes a "projected quasi-Newton method" (see Gill, Murray and Wright, 1981). In general, the sequence of iterates will differ from that generated by the SQP method because only one linear constraint enters or leaves the working set during each major iteration. This method is likely to be efficient on problems in which relatively *many* constraints are active at the solution. If the initial working set of bounds and linear constraints is large, curvature information can be accumulated in R_L without the risk of deleting too many constraints during a single major iteration. Under these circumstances, the matrix $Z_L^T H Z_L$ may be updated with a positive-definite quasi-Newton approximation when the full H cannot.

8. Representation of the Hessian in sparse quadratic programs

All the complexities of implementation mentioned earlier are magnified when developing methods for large-scale optimization. To indicate the shift in perspective, we consider an SQP method for solving nonlinearly constrained problems in which the Hessian of the Lagrangian is sparse and known exactly. Thus in each QP subproblem, H is available in some form that allows the quadratic objective and its gradient q ($= Hp + g$) to be computed.

Recall that, in the QP phase, the search direction is taken as Zp_Z, where p_Z satisfies

$$Z^T H Z p_Z = -Z^T q. \tag{15}$$

The first issue in solving (15) is the representation of Z. Although methods for computing an orthogonal basis Z for the null space of a sparse matrix have been the subject of much recent research (see, e.g., Coleman and Pothen, 1984), they are not practical for sparse quadratic programs because of the need to *update* Z at every iteration of the QP method. Instead, methods have been developed in which the matrix Z is not stored explicitly.

Assume that the n_{FR} columns of the working set C are ordered so that

$$C = (B \quad S),$$

where the $m \times m$ matrix B is non-singular. (In practice, the columns of B may occur anywhere in C.) The matrix Z defined by

$$Z = \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix} \tag{16}$$

provides a basis for the null space of C, and is called the *reduced-gradient* form of the null space (see, e.g., Murtagh and Saunders, 1978). This form is effective for sparse problems because operations with Z and Z^T may be performed using a factorization of the sparse matrix B. Let s denote $n_{FR} - m$ (the number of columns of S). The reduced-gradient form of Z can been used with great success in quasi-Newton methods for sparse linearly constrained problems in which s is small, by maintaining a dense Cholesky factorization of a quasi-Newton approximation to the projected Hessian (see Murtagh and Saunders, 1978).

In the case of sparse quadratic programming, a factorization of the projected Hessian matrix Z^THZ is needed to solve (15). In a general dense QP method, the initial projected Hessian would simply be formed at the start of the QP phase by multiplying the explicit matrices Z and H. One might suppose that this procedure could carry over to the sparse case, assuming that s is small enough so that the explicit projected Hessian can be stored. (Note that the projected Hessian will generally be dense, even if H and B are sparse.)

Unfortunately, even if s is small, *forming* the explicit initial projected Hessian may involve a substantial amount of work. When Z is defined by (16), computation of Z^THZ requires the solution of $2s$ systems of size $m \times m$. For this reason, if the number of iterations in the QP method is small, computation of the *first* search direction will dominate the time required to solve the quadratic program. (This situation always applies during the last few major iterations of an SQP method.) To put the relative costs into perspective, note that computation of a single row and column of Z^THZ requires approximately the same amount of work as a single iteration of the simplex method (i.e., two linear systems of order m). If $s = 100$, forming the initial projected Hessian would require the equivalent of 100 iterations of the simplex method!

An alternative approach is to use a *quasi-Newton method to solve the quadratic program.* The required quasi-Newton approximation to Z^THZ is maintained using the change in p and q between successive minor iterates. (H is needed only to form Hp, from which q and the QP objective function are easily obtained.)

It may not be obvious why this is an improvement. If the exact projected Hessian is not used, the resulting search direction is no longer the step to the minimum of the quadratic function in the subspace defined by Z. In effect, the cost of forming the initial projected Hessian seems merely to have been spread over a number of iterations, since at least s iterations should be required to produce the "true" projected Hessian. However, in the SQP context, the major gain is that the Cholesky factor of the projected Hessian at the solution of one quadratic program can be used to initiate the solution of the next. This approach has proved to be very successful in the implementation of an SQP method for solving large-scale problems arising in the optimal distribution of electrical power (see Burchett, Happ and Vierath, 1984).

This is another situation in which changes to the initialization procedure ultimately produce a method which is quite different from the original. During early iterations, significantly different working sets are generated by the quasi-Newton quadratic program because the working set is usually altered well before the approximate projected Hessian has any resemblance to the exact projected Hessian.

References

Biggs, M. C. (1972). "Constrained minimization using recursive equality quadratic programming", in *Numerical Methods for Non-Linear Optimization* (F. A. Lootsma, ed.), pp. 411 428, Academic Press, London and New York.

Burchett, R. C., Happ, H. H. and Vierath, D. R. (1984). Quadratically convergent optimal power flow, presented at the IEEE/PES 1984 Winter Meeting, Dallas, Texas (to appear in *IEEE Transactions on Power Apparatus and Systems*).

Byrd, R. H. and Schnabel R. B. (1984). Continuity of the null space basis and constrained optimization, Report CU-CS-272-84, Department of Computer Science, University of Colorado, Boulder, Colorado.

Cody, W. J. (1974). The construction of numerical subroutine libraries, *SIAM Review* **16**, pp. 36-46.

Coleman, T. F. and Conn, A. R. (1982). Nonlinear programming via an exact penalty function, *Math. Prog.* **24**, pp. 123 161.

Coleman, T. F. and Pothen, A. (1984). The sparse null space basis problem, Report 84-598, Department of Computer Science, Cornell University, Ithaca, New York.

Cowell, W. R. (ed.) (1983). *Sources and Development of Mathematical Software*, Prentice-Hall, Englewood Cliffs, New Jersey.

Dennis, J. E., Jr. and Schnabel, R. E. (1981). "A new derivation of symmetric positive definite secant updates", *Nonlinear Programming 4* (O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds.), pp. 167 199, *Academic Press, London and New York.*

Dennis, J. E., Jr. and Schnabel, R. B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Gabay, D. (1982). Reduced quasi-Newton methods with feasibility improvement for nonlinearly constrained optimization, *Math. Prog. Study* **16**, pp. 18-44.

Gill, P. E., Golub, G. H., Murray, W. and Saunders, M. A. (1974). Methods for modifying matrix factorizations, *Mathematics of Computation* **28**, pp. 505 535.

Gill, P. E. and Murray, W. (1974). "Quasi-Newton methods for linearly constrained optimization", in *Numerical Methods for Constrained Optimization* (P. E. Gill and W. Murray, eds.), pp. 67 92, Academic Press, London and New York.

Gill, P. E. and Murray, W. (1978). Numerically stable methods for quadratic programming, *Math. Prog.* **14**, pp. 349 372.

Gill, P. E., Murray, W., Picken, S. M. and Wright, M. H. (1979). The design and structure of a Fortran program library for optimization, *ACM Transactions on Mathematical Software* **5**, pp. 259 283.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984*a*). User's guide for QPSOL (Version 3.2): a Fortran package for quadratic programming, Report SOL 84-6, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984*b*). User's guide for NPSOL (Version 2.1): a Fortran package for nonlinear programming, Report SOL 84-7, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984*c*). Procedures for optimization problems with a mixture of bounds and general linear constraints, *ACM Transactions on Mathematical Software* **10**, pp. 282–298.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1985*a*). The design and implementation of a quadratic programming algorithm, to appear, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A., Stewart, G. W. and Wright, M. H. (1985*b*). Properties of a representation of a basis for the null space, Report SOL 85-1, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W. and Wright, M. H. (1981). *Practical Optimization*, Academic Press, London and New York.

Han, S.-P. (1976). Superlinearly convergent variable metric algorithms for general nonlinear programming problems, *Math. Prog.* **11**, pp. 263–282.

Murray, W. and Wright, M. H. (1978). Methods for nonlinearly constrained optimization based on the trajectories of penalty and barrier functions, Report SOL 78 23, Department of Operations Research, Stanford University.

Murtagh, B. A. and Saunders, M. A. (1978). Large-scale linearly constrained optimization, *Math. Prog.* **14**, pp. 41–72.

Nocedal, J. and Overton, M. (1983). Projected Hessian updating algorithms for nonlinearly constrained optimization, Report 95, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York.

Powell, M. J. D. (1977). A fast algorithm for nonlinearly constrained optimization calculations, *Report DAMTP 77/NA 2, University of Cambridge, England.*

Powell, M. J. D. (1983). "Variable metric methods for constrained optimization", in *Mathematical Programming: The State of the Art*, (A. Bachem, M. Grötschel and B. Korte, eds.), pp. 288–311, Springer-Verlag, Berlin, Heidelberg, New York and Tokyo.

Robinson, S. M. (1974). Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms, *Math. Prog.* **7**, pp. 1–16.

Stewart, G. W. (1973). *Introduction to matrix computations*, Academic Press, London and New York.

Wilson, R. B. (1963). *A Simplicial Algorithm for Concave Programming*, Ph.D. Thesis, Harvard University.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARO 21592.4-MA	2. GOVT ACCESSION NO. NA AD-A151 480	3. RECIPIENT'S CATALOG NUMBER NA
4. TITLE (and Subtitle) SOFTWARE AND ITS RELATIONSHIP TO METHODS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip E. Gill, Walter Murray, Michael A. Saunders and Margaret H. Wright		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0267 DAAG29-84-K-0156
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-143
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE November 1984 13. NUMBER OF PAGES 15
U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Numerical Software
Numerical Analysis
Optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

One view of numerical software is that it is simply a computer implementation of a known method. Implicit in this view is the assumption that the flow of information is in one direction only. However, developments in methods and software are intimately related, and neither is complete if considered in isolation. In this paper, we illustrate how the development of numerical software has influenced our research in optimization methods.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END

FILMED

4-85

DTIC